Bayesian Hierarchical MPT Models

Application and Practice with TreeBUGS

Daniel W. Heck

2018-09-11

TreeBUGS: Bayesian Hierarchical MPT Modeling

1. The R package `TreeBUGS`
2. Basic modeling
   - Model fitting
   - Convergence
   - Plots
   - Model fit
3. Advanced modeling
   - Within-subject comparisons
   - Between-subject comparisons
   - Covariates
4. Sensitivity/robustness analysis
   - Priors
   - Predictive distributions
   - Simulation

Software for Hierarchical MPTs

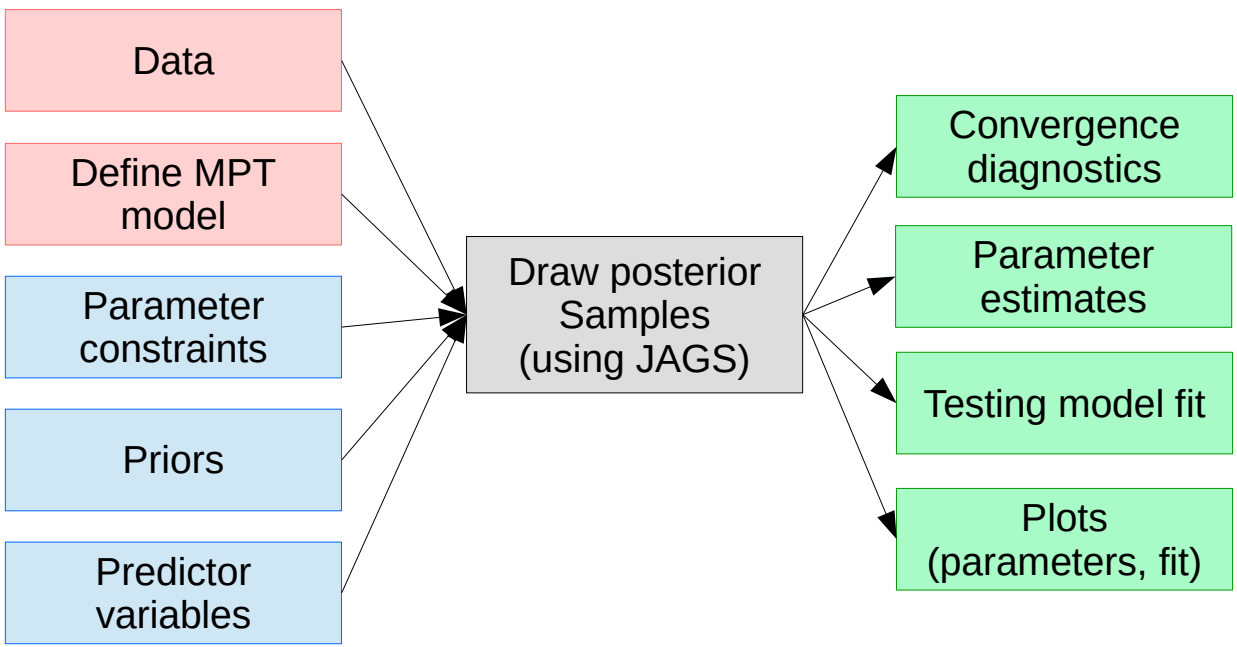
- Implementation of MCMC sampling in R/C/Fortran (Klauer, 2010)
- General-purpose software: WinBUGS/JAGS/Stan (Matzke et al., 2015)
- Requires re-implementation of summaries, statistics, plots

TreeBUGS: A user-friendly R package (Heck, Arnold, & Arnold, 2018)

- Easy-to-use, open source, free
- Fitting and testing MPT models
  - Posterior sampling, summary statistics, and plots
  - Data generation, robustness simulations
  - Change priors, add predictors, etc.
  - Current limitation: Crossed random effects (persons + items)

Functionality of TreeBUGS

- Input: R objects or text/csv files (minimal R knowledge required)
- Priors and other details can be changed in R
- TreeBUGS translates the model to JAGS (Plummer, 2003) to draw posterior samples
- Functions for post-processing, summaries and plots



Reference

TreeBUGS Paper

Behav Res
DOI 10.3758/s13428-017-0869-7

CrossMark

TreeBUGS: An R package for hierarchical multinomial-processing-tree modeling

Daniel W. Heck[1] · **Nina R. Arnold**[1] · **Denis Arnold**[2,3]

Abstract Multinomial processing tree (MPT) models are a class of measurement models that account for categorical data by assuming a finite number of underlying cognitive processes. Traditionally, data are aggregated across participants and estimates, fit statistics, and within- and between-subjects comparisons, as well as goodness-of-fit and summary plots. We also propose and implement novel statistical extensions to include continuous and discrete predictors (as either fixed or

Basic Modeling

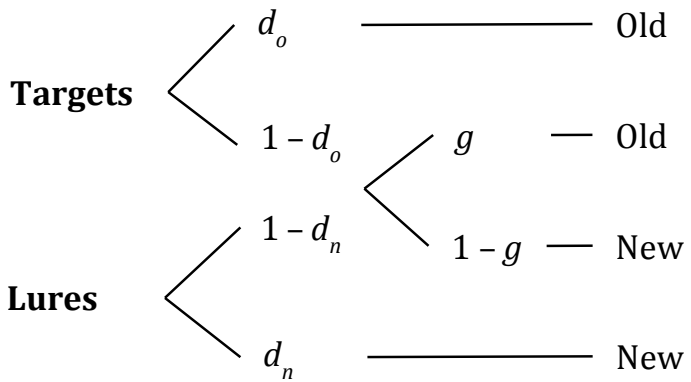
(corresponding R script: `08-ApplicationIV-TreeBUGS.R`)

MPT Model Specification

- MPT structure is defined in an EQN model file
  - Simple: copy from multiTree :-)
- Main differences
  - the symbol # allows to add comments
  - you may use numbers for equality constraints (e.g., replace g by `0.50`)

Two-high threshold model
(file: `2htm.eqn`)

```
# Targets
target  hit  do
target  hit  (1-do)*g
target  miss (1-do)*  (1-g)

# Lures
lure  cr  dn
lure  fa  (1-dn)*g
lure  cr  (1-dn)*(1-g)
```



Data Structure

- Data: Response frequencies in wide format
  - One line per person
  - One category per column
  - Column names must be identical to the EQN categories!
- Either supplied in .csv-file or as `data.frame` / `matrix` in R

Example: `2htm.csv`

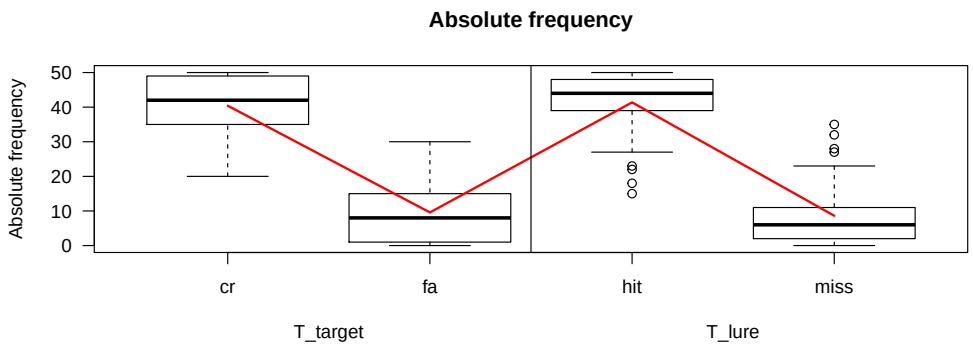
```
frequencies <- read.csv("2htm.csv")
head(frequencies, 5)
```

```
##   cr fa hit miss
## 1 49  1  49    1
## 2 46  4  39   11
## 3 43  7  45    5
## 4 49  1  50    0
## 5 43  7  48    2
```

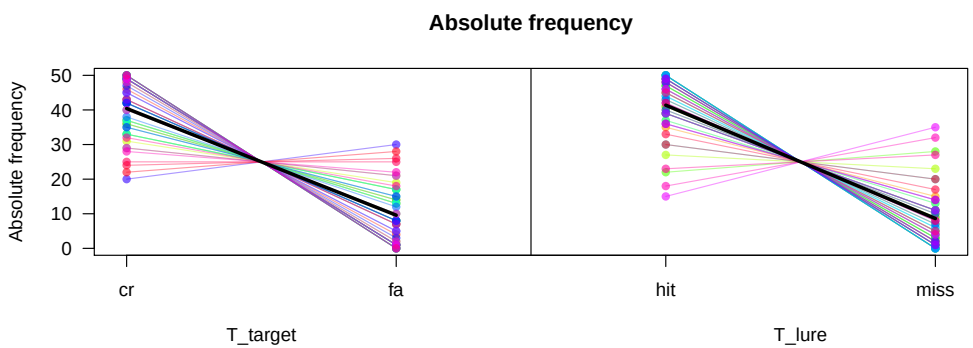
Heterogeneity

- Load TreeBUGS and plot heterogeneity
- If response frequencies are homogeneous, standard (fixed-effects) MPT models are statistically more efficient

```
library("TreeBUGS")
plotFreq(frequencies, eqnfile = "2htm.eqn")
```




```
plotFreq(frequencies, boxplot = FALSE, eqnfile = "2htm.eqn")
```



Fitting MPT Models

- Fitting an MPT model in TreeBUGS
  - Model: Text file in EQN syntax (with model equations)
  - Data: `.csv`file
  - Constraints: text file with equality contraints

```
fit <- traitMPT(eqnfile = "htm.txt",
                data = "responses.csv",
                restrictions = "2htm_constraints.txt")

# beta-MPT: different function, but identical arguments
fit_beta <- betaMPT(eqnfile = "htm.txt",
                    data = "responses.csv",
                    restrictions = "2htm_constraints.txt")
```

Fitting an MPT Model in R

- Alternative: define everything directly in R
  - Model: Text string (`character` in apostrophes)
  - Data: Matrix or data frame
  - Constraints: A list

```
htm <- "
target  hit  do
target  hit  (1-do)*g
target  miss (1-do)*  (1-g)

lure  cr  dn
lure  fa  (1-dn)*g
lure  cr  (1-dn)*(1-g)
"
fit <- traitMPT(eqnfile = htm,
                data = frequencies,
                restrictions = list("dn=do", "g=.50"))
```

Parameter Constraints

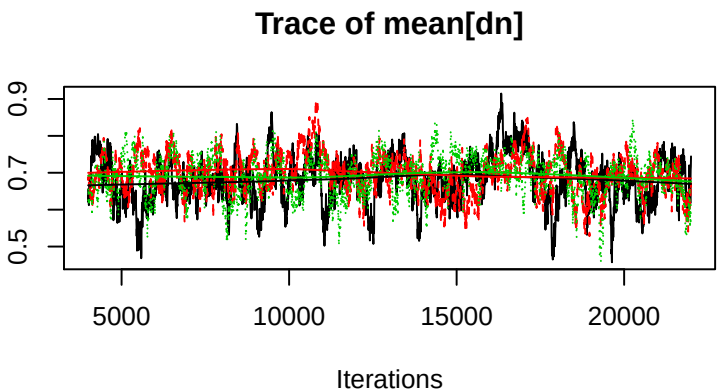
Equality constraints

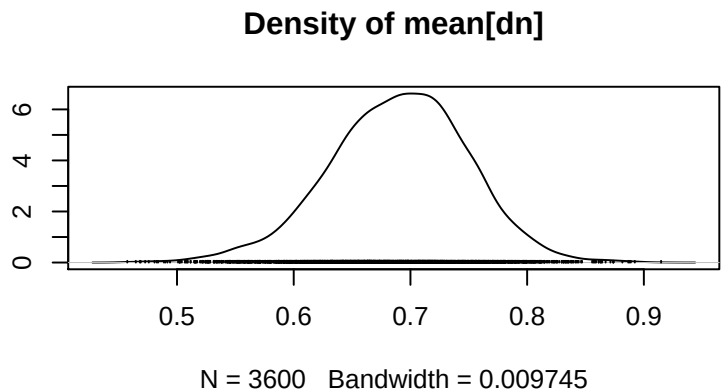
```
# (A) use a general model file and constrain parameters:
fit <- traitMPT(eqnfile = htm,
                data = frequencies,
                restrictions = list("dn=do", "g=.50"))

# (B) hard-coding of constraints in the EQN file:
htm_constr <- "
target  hit  d
target  hit  (1-d)*.50
target  miss (1-d)*.50

lure  cr  d
lure  fa  (1-d)*.50
lure  cr  (1-d)*.50
"
fit <- traitMPT(eqnfile = htm_constr,
                data = frequencies)
```

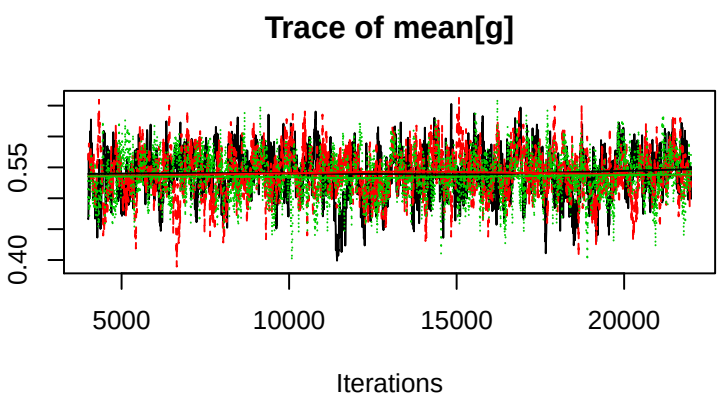
Convergence

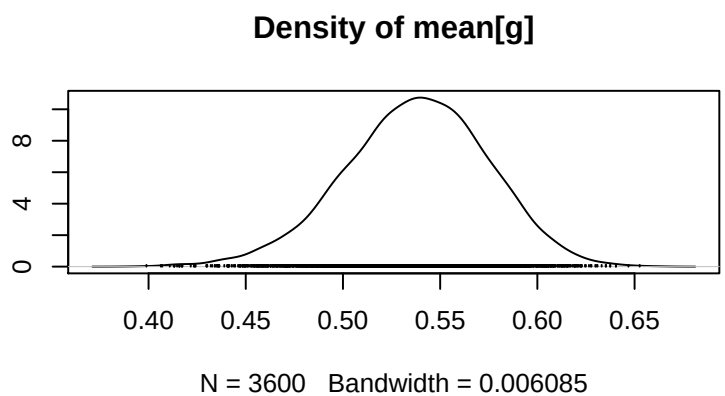
- Convergence check
  - Posterior/ MCMC samples should look unsystematic (like a hairy caterpillar)
  - For more options, see: `?plot.traitMPT`

```
plot(fit, parameter = "mean", type = "default")
```





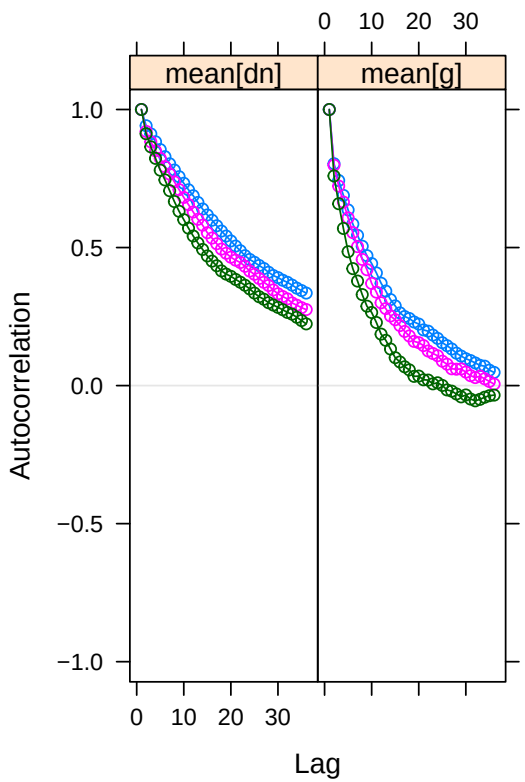




Autocorrelation function

- How strongly are the MCMC samples correlated between iteration t and iteration $t + \text{Lag}$?
- Ideally, these curves should rapidly decrease towards zero.

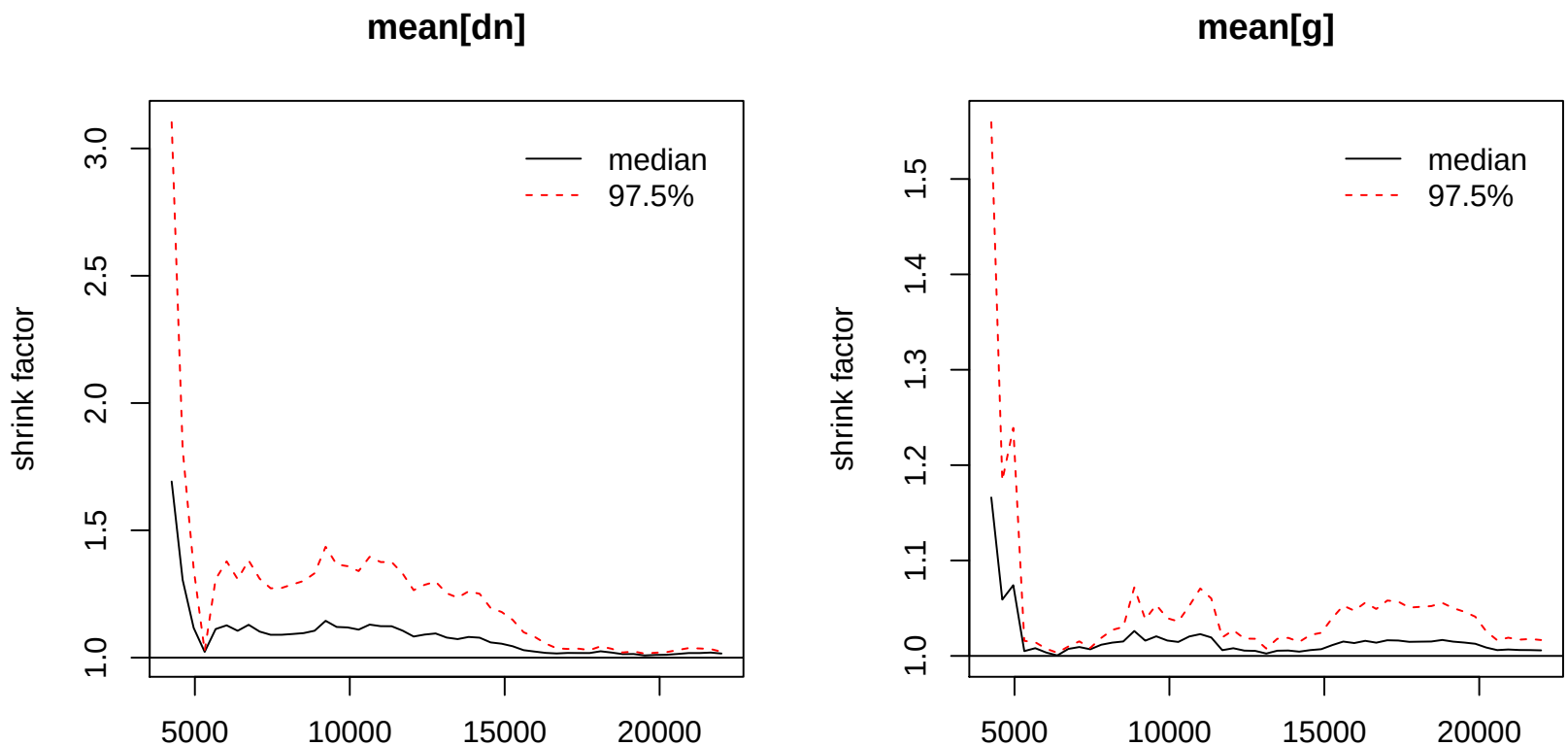
```
plot(fit, parameter = "mean", type = "acf")
```



Convergence

- Plot evolution of Gelman-Rubin statistic
  - Also known as: "potential scale reduction factor" or "R hat"
  - Similar to ANOVA: Compares between-chain and within-chain variances (large differences between these variances indicate nonconvergence)
  - Statistic should be close to 1

```
plot(fit, parameter = "mean", type = "gelman")
```



Options for MCMC Sampling

- If the model has not converged, it must be fitted with more conservative settings:

```
fit <- traitMPT(
  eqnfile = htm, data = frequencies,
  restrictions = list("dn=do"),

  n.adapt = 5000, # longer adaption of JAGS increases efficiency of sampling
  n.burnin = 5000,# longer burnin avoids issues due to bad starting values
  n.iter = 30000, # drawing more MCMC samples leads to higher precision
  n.thin = 10,    # ommitting every 10th sample reduces memory load
  n.chains = 4)   # more MCMC chains increase precision
```

```
## MCMC sampling started at  2018-09-10 16:48:45
## Calling 4 simulations using the parallel method...
## Following the progress of chain 1 (the program will wait for all
## chains to finish before continuing):
## Welcome to JAGS 4.3.0 on Mon Sep 10 16:48:49 2018
## JAGS is free software and comes with ABSOLUTELY NO WARRANTY
## Loading module: basemod: ok
## Loading module: bugs: ok
## . Loading module: dic: ok
## . Loading module: glm: ok
## . . Reading data file data.txt
## . Compiling data graph
## . . Resolving undeclared variables
```

Extend MCMC Sampling

- If the MCMC samples are OK but higher precision is needed:
  - Extend sampling and add new MCMC samples to the fitted JAGS object

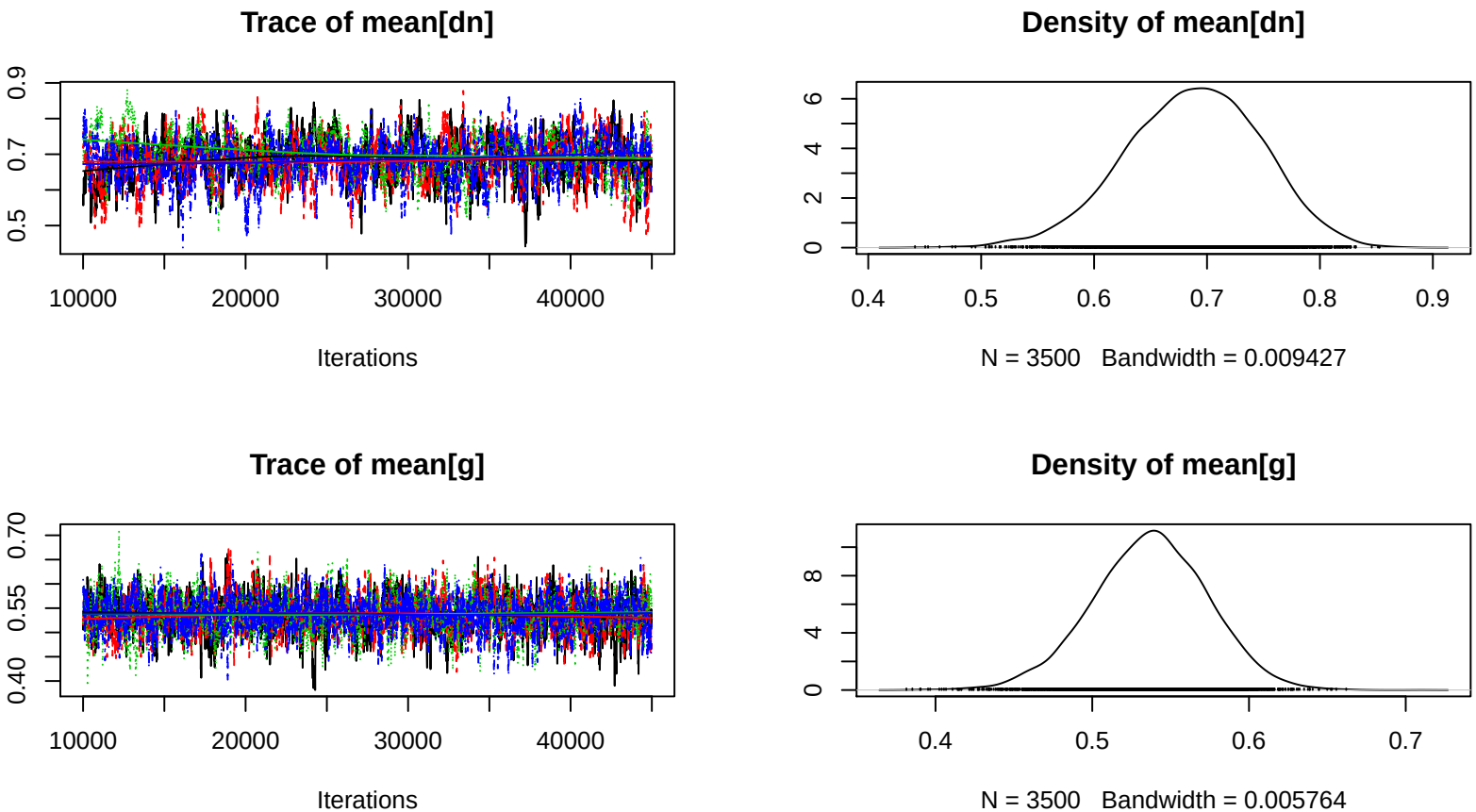
```
fit2 <- extendMPT(fit,                # fitted MPT model
                  n.adapt = 2000,     # JAGS need to restart and adapt again
                  n.burnin = 0,       # burnin not needed if previous samples are OK
                  n.iter = 10000)     # how many additional iterations?
```

```
## Calling 4 simulations using the parallel method...
## Following the progress of chain 1 (the program will wait for all
## chains to finish before continuing):
## Welcome to JAGS 4.3.0 on Mon Sep 10 16:49:30 2018
## JAGS is free software and comes with ABSOLUTELY NO WARRANTY
## Loading module: basemod: ok
## Loading module: bugs: ok
## . Loading module: dic: ok
## . Loading module: glm: ok
## . . Reading data file data.txt
## . Compiling data graph
##    Resolving undeclared variables
##    Allocating nodes
##    Initializing
##    Reading data back into data table
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 100
##    Unobserved stochastic nodes: 55
```

Convergence for Second Fit

- Check convergence again:

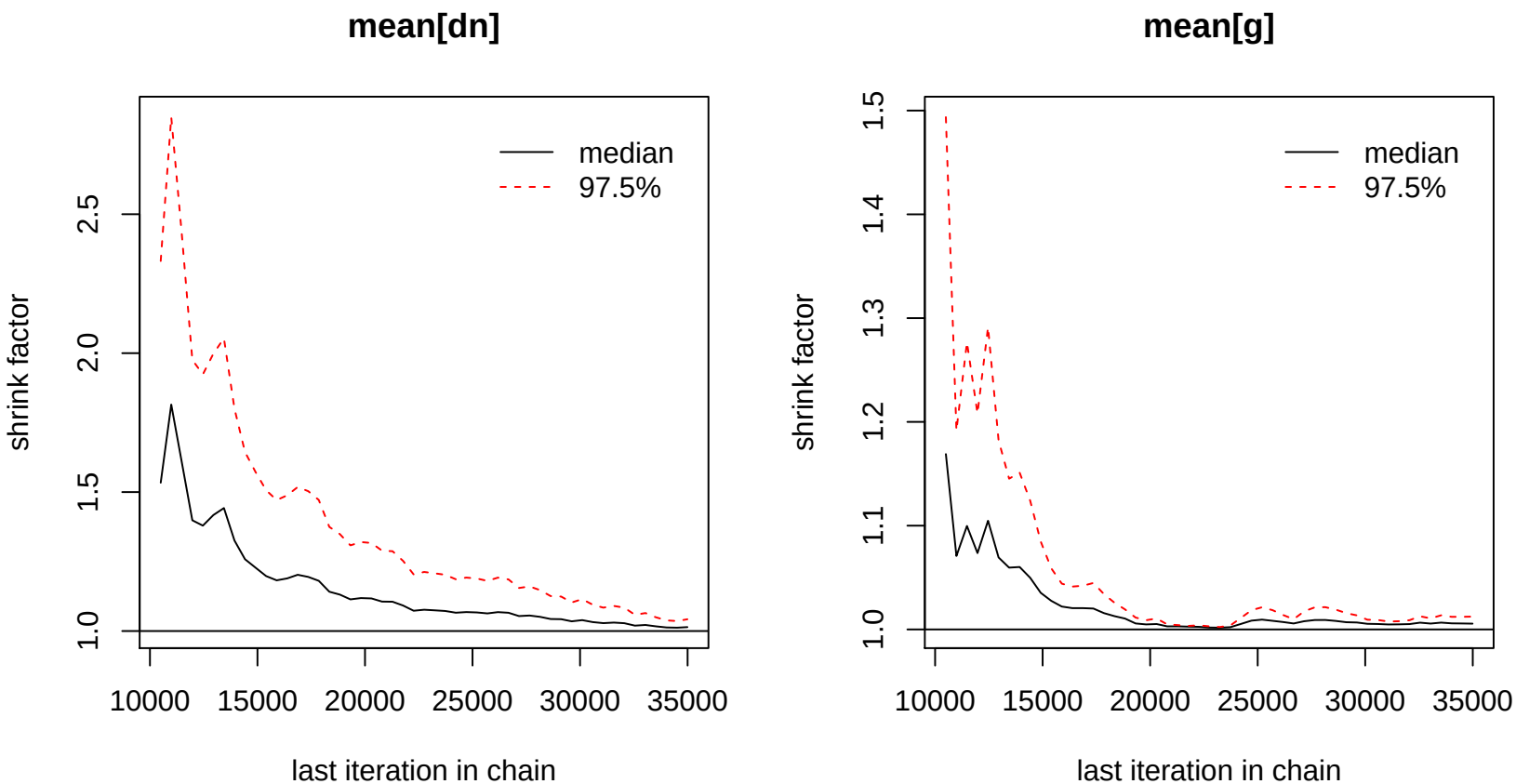
```
plot(fit2, parameter = "mean", type = "default")
```



Convergence for Second Fit

- Check convergence again:

```
plot(fit, parameter = "mean", type = "gelman")
```



Parameter Estimates

- Summary statistics for posterior distribution:
  - Posterior mean and median (50% quantile)
  - Posterior standard deviation (SD, similar to standard error)
  - Bayesian credibility interval (2.5% and 97.5% quantiles)

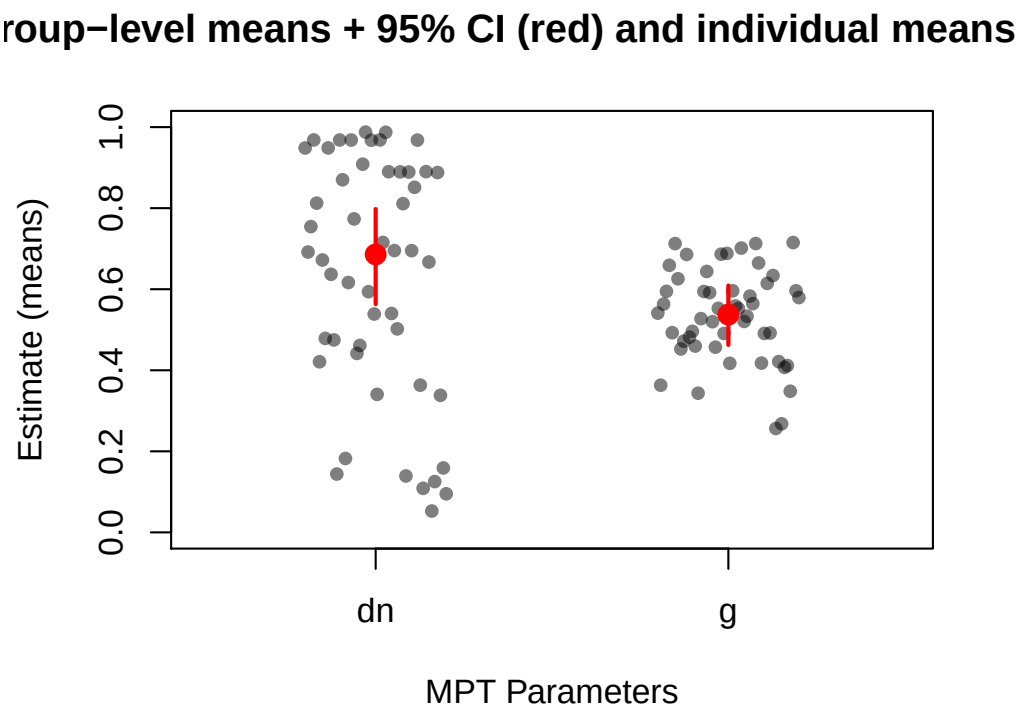
```
summary(fit2)
```

```
## Call:
## [[1]]
## traitMPT(eqnfile = htm, data = frequencies, restrictions = list("dn=do"),
##     n.iter = 30000, n.adapt = 5000, n.burnin = 5000, n.thin = 10,
##     n.chains = 4)
##
## [[2]]
## extendMPT(fittedModel = fit, n.iter = 10000, n.adapt = 2000,
##     n.burnin = 0)
##
##
## Group-level medians of MPT parameters (probability scale):
##          Mean    SD  2.5%   50% 97.5% Time-series SE n.eff  Rhat R_95%
## mean_dn 0.687 0.060 0.563 0.690 0.798          0.002   628 1.007 1.015
## mean_g  0.537 0.037 0.461 0.538 0.608          0.001  1501 1.003 1.007
##
## Mean/Median of latent-trait values (probit-scale) across individuals:
##                Mean    SD   2.5%   50% 97.5% Time-series SE n.eff  Rhat
## latent_mu_dn 0.496 0.172  0.160 0.495 0.833          0.007   624 1.006
## latent_mu_g  0.093 0.094 -0.098 0.094 0.273          0.002  1501 1.003
##              R_95%
## latent_mu_dn 1.014
## latent_mu_g  1.007
##
```

Plot Parameter Estimates

- Estimates for group-level parameters
  - Overall mean μ: Posterior mean and Bayesian credibility interval
  - Individual parameters θ_i: Posterior mean

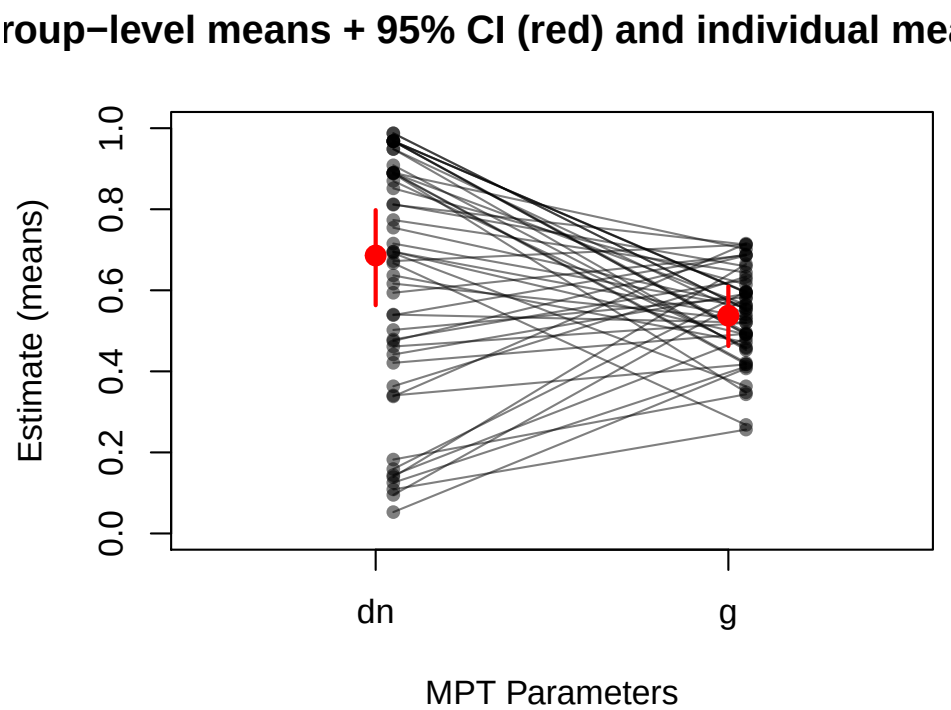
```
plotParam(fit)
```



Plot Parameter Estimates

- Plot parameter profiles
  - E.g., assess test-retest reliability of a parameter (Michalkiewicz & Erdfelder, 2016)
- For more options, see: `?plotParam`

```
plotParam(fit, addLines = TRUE, select = c("dn", "g"))
```



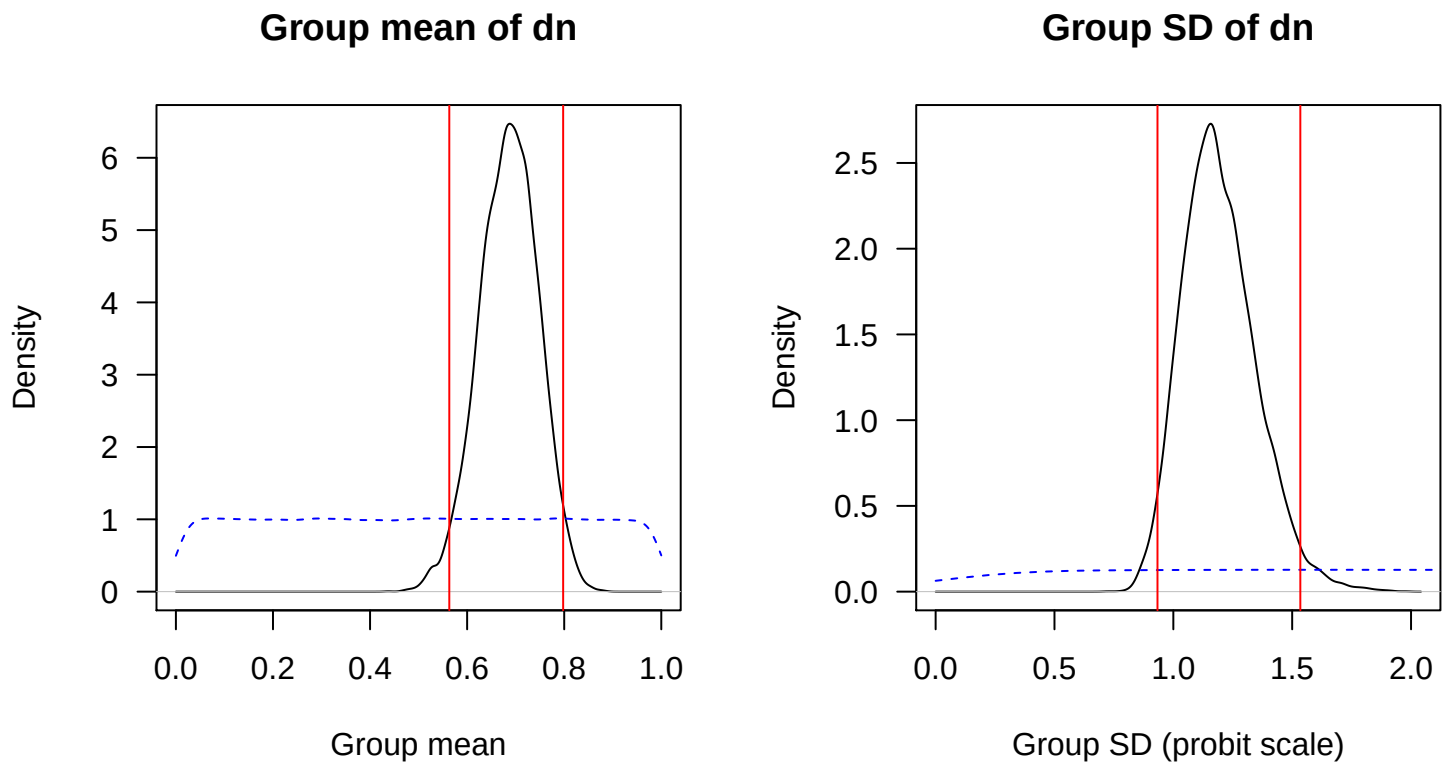
Compare Prior and Posterior

How much did we learn about the parameters?

- Graphical assessment: Plot prior (blue) and posterior (black) densities

```
plotPriorPost(fit)
```

```
## Press <Enter> to show the next plot.
```

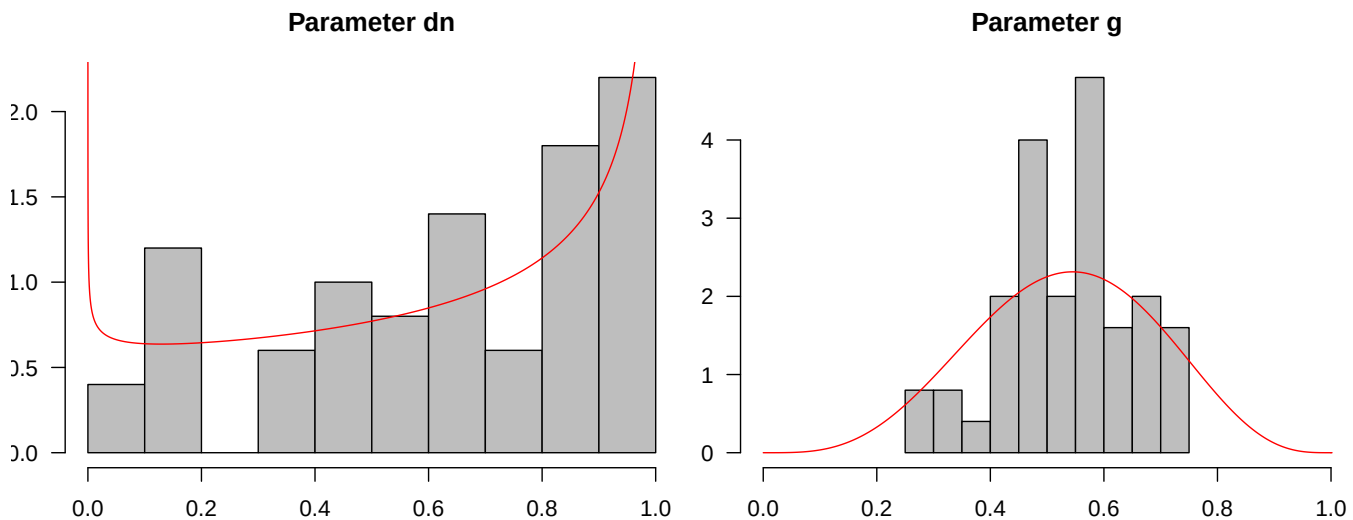


Group-Level Distribution

Distribution of individual estimates

- Histogram: Distribution of θ estimates (posterior mean per person)
- Red density: Estimated group-level distribution

```
plotDistribution(fit)  # graphical test
```

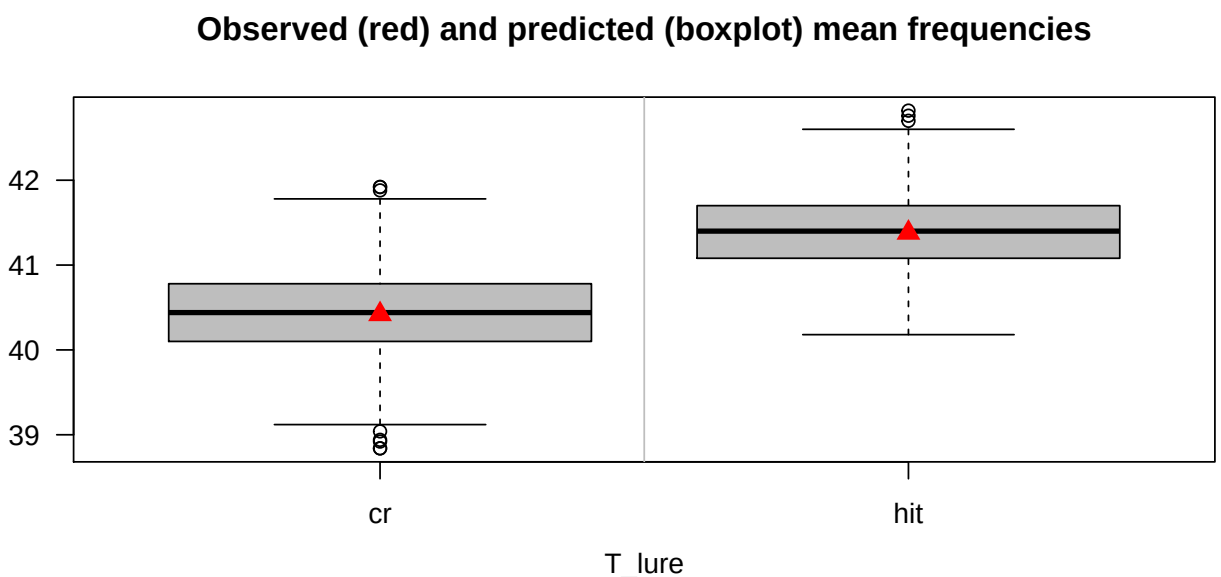


Model Fit: Predicted vs. Observed Data

Graphical test of model fit

- Plot means of observed frequencies
- Compare against posterior-predicted frequencies (boxplots)

```
plotFit(fit)  # graphical test
```



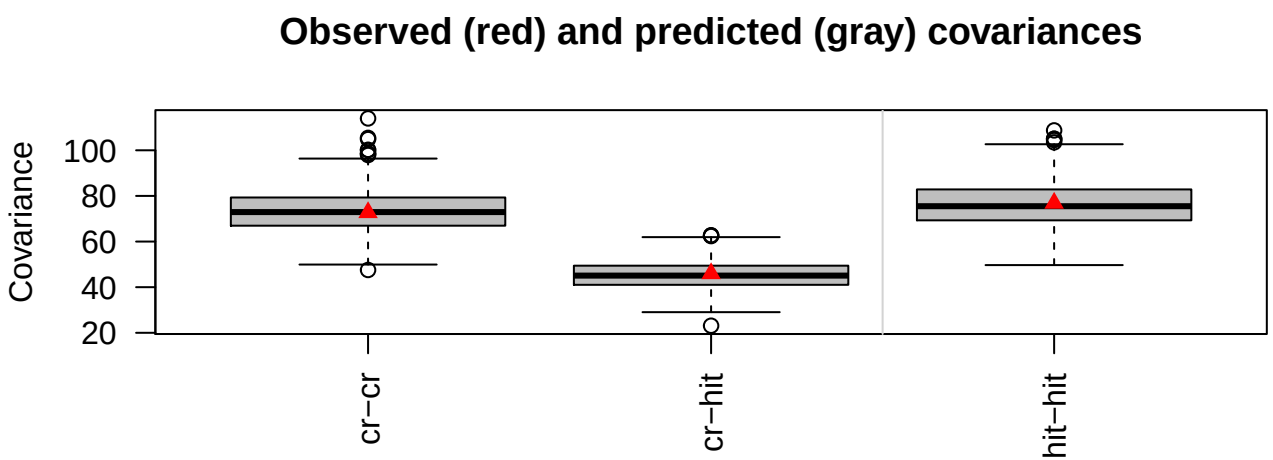
Model Fit: Predicted vs. Observed Data

- Model fit for *covariance* of observed frequencies

```
cov(frequencies)            # this is the observed covariance matrix we test
```

```
##              cr        fa       hit      miss
## cr    72.73837 -72.73837  45.98000 -45.98000
## fa   -72.73837  72.73837 -45.98000  45.98000
## hit   45.98000 -45.98000  76.85265 -76.85265
## miss -45.98000  45.98000 -76.85265  76.85265
```

```
plotFit(fit, stat = "cov")  # graphical test
```



Model Fit: Predicted vs. Observed Data

Testing model fit

- The statistics T1 and T2 quantify the discrepancy between observed and expected means/covariances (similar to Pearson's X^2)
- Posterior predictive p-values
  - Values around .50 indicate good model fit
  - Values close to 0 (or close to 1) indicate misfit
  - In contrast to frequentist p-values, PPP values are *not* uniformly distributed when generating data from the correct model

```
PPP(fit, M = 2000, nCPU = 4)
```

```
##  ## Mean structure (T1):
##  Observed =  0.03169322 ; Predicted =  0.03175106 ; p-value =  0.5045
##
## ## Covariance structure (T2):
##  Observed =  6.572722 ; Predicted =  7.430602 ; p-value =  0.5295
##
## ## Individual fit (T1):
##     1     2     3     4     5     6     7     8     9    10    11    12
## 0.578 0.450 0.522 0.418 0.478 0.516 0.418 0.518 0.270 0.526 0.501 0.524
##    13    14    15    16    17    18    19    20    21    22    23    24
## 0.372 0.503 0.513 0.532 0.407 0.518 0.502 0.520 0.442 0.220 0.486 0.358
##    25    26    27    28    29    30    31    32    33    34    35    36
## 0.500 0.510 0.441 0.552 0.222 0.238 0.526 0.538 0.536 0.557 0.366 0.516
##    37    38    39    40    41    42    43    44    45    46    47    48
## 0.389 0.538 0.512 0.348 0.508 0.483 0.422 0.308 0.318 0.543 0.192 0.464
##    49    50
```

Summary

Modeling with TreeBUGS is simple

1. Define model and clean data
2. Draw MCMC samples
3. Check convergence
4. Check model fit
5. Interpret/plot parameters

Note that these are the usual steps in any Bayesian analysis...

Advanced Modeling

Assessing within-subject differences in parameters

1. Data: Additional columns for separate within-subject conditions
2. Model: Write EQN file for within-subject design
3. MCMC sampling (as usual)
4. Comparison: Compute differences of parameters (transformed parameters)

(1) Data structure for within-subject design

```
freq_within <- read.csv("2htm_within.csv")
head(freq_within, 3)
```

```
##   high_cr high_fa high_hit high_miss low_cr low_fa low_hit low_miss
## 1      47       3       47         3     33     17      27       23
## 2      43       7       47         3     37     13      31       19
## 3      38      12       40        10     36     14      39       11
```

(2) Model: Function for writing within-subject EQN files

- TreeBUGS has a function extends a standard MPT model to multiple within-subject conditions
- Essentially, model equations are copied and each parameter gets a new label (e.g., `d_condition1`)

```
# create EQN file for within-subject manipulations
withinSubjectEQN(htm_d,
                 labels = c("high","low"),  # factor labels
                 constant=c("g"))           # constant parameters
```

```
##           Tree  Category         Equation
## 1  high_target  high_hit           d_high
## 2  high_target  high_hit     (1-d_high)*g
## 3  high_target high_miss (1-d_high)*(1-g)
## 4    high_lure   high_cr           d_high
## 5    high_lure   high_fa     (1-d_high)*g
## 6    high_lure   high_cr (1-d_high)*(1-g)
## 7   low_target   low_hit            d_low
## 8   low_target   low_hit      (1-d_low)*g
## 9   low_target  low_miss  (1-d_low)*(1-g)
## 10    low_lure    low_cr            d_low
## 11    low_lure    low_fa      (1-d_low)*g
## 12    low_lure    low_cr  (1-d_low)*(1-g)
```

(4) Transformed parameters

- Often the interest is in the difference of a parameter across conditions
  - Example: Difference in memory strength $\Delta_d = d_{\text{high}} - d_{\text{low}}$
- Based on the MCMC samples, we can simply compute any function of interest
  - We get a new set of posterior samples that can be summarized as usual

```
# fit to all conditions:
fit_within <- traitMPT("2htm_within.eqn", "2htm_within.csv")
```

```
# compute difference in d:
diff_d <- transformedParameters(
  fit_within,
  transformedParameters = list("diff_d = d_high - d_low"),
  level = "group")
summary(diff_d)$statistics
```

```
##           Mean             SD       Naive SE Time-series SE
##   0.2973143953   0.0384344293   0.0003698355   0.0013866989
```

Comparisons in between-subject designs

1. Fit MPT model to each condition separately
   - Separate group-level parameters μ and Σ per group
2. Compute differences in group-level parameters across conditions/models

```
fit1 <- traitMPT(htm_d, "2htm.csv")
fit2 <- traitMPT(htm_d, "2htm_group2.csv")
```

```
diff_between <- betweenSubjectMPT(
                    fit1, fit2,             # fitted MPT models
                    par1 = "d",             # parameter to test
                    stat = c("x-y","x>y"))  # transformed parameters)
diff_between
```

```
##             Mean    SD  2.5%   50% 97.5% Time-series SE n.eff  Rhat R_95%
## d.m1-d.m2 0.389 0.074 0.241 0.389 0.534          0.004   314 1.008 1.025
## d.m1>d.m2 1.000 0.000 1.000 1.000 1.000            NaN     0   NaN   NaN
```

Covariates: Correlations

Correlation of MPT parameters with external covariates

- New argument `covData`: A data frame or file name with values of the covariate(s)
- TreeBUGS computes the correlation of these covariates with the latent person parameters θ (probit values)

```
fit_cor <- traitMPT(htm_d, data = "2htm.csv",
                    covData = "covariates.csv")   # data with covariate(s)
```

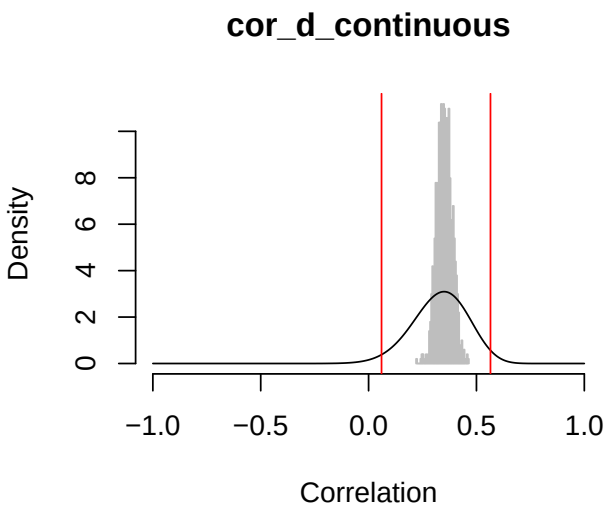
```
# uncertainty with respect to the MPT parameter estimates:
round(fit_cor$summary$group$cor[,-6], 2)
```

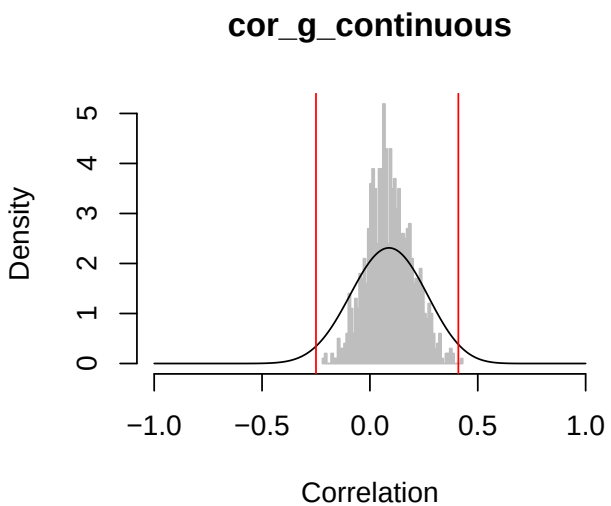
```
##                   Mean   SD  2.5%  50% 97.5% n.eff Rhat R_95%
## cor_d_continuous 0.35 0.04  0.28 0.35  0.42  5635    1     1
## cor_g_continuous 0.09 0.11 -0.11 0.09  0.30  1266    1     1
```

Covariates: Correlations

- Note that the posterior samples of the (descriptive) correlations only reflect uncertainty with respect to the MPT parameters
- We also need to consider the number of participants (sample size)!
- Solution: Use an analytical solution or the posterior distribution of the correlation (Ly et al., 2018)

```
correlationPosterior(fit_cor)
```






```
##                    2.5%  50% 97.5%
## cor_d_continuous   0.06 0.33 0.565
## cor_g_continuous -0.25 0.08 0.410
```

Covariates: Regression

Regression of MPT parameters on covariates

- Example: Predict memory performance d as a function of age
- Statistically, this requires an regression extension to the model
- The latent probit values θ_i' are predicted by a design matrix X:

$$\theta_i' = \mu + X_i\beta + \delta_i$$

Implementation in TreeBUGS

- Requires only two new arguments to provide the data (age of persons) and the regression structure (predict parameter D_n by age)

```
fit_regression <- traitMPT(htm_d, data = "2htm.csv",
                           covData = "covariates.csv",
                           predStructure = list("d ; continuous"))
```

```
round(fit_regression$summary$group$slope[,-6], 2)
```

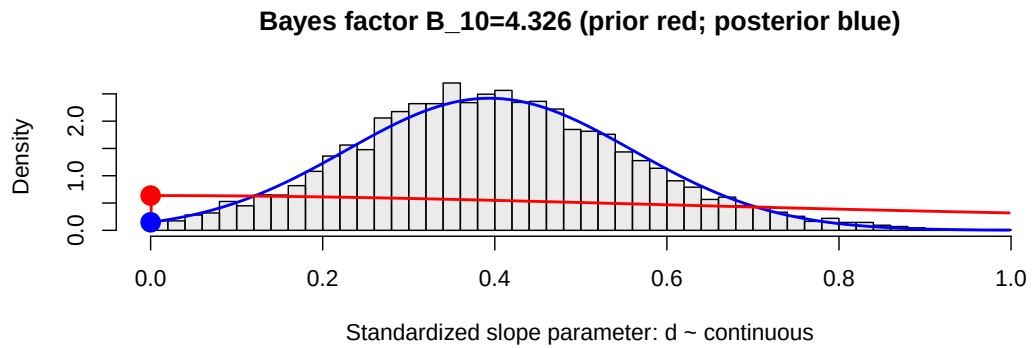
```
##   Mean     SD   2.5%    50%  97.5%  n.eff   Rhat  R_95%
##   0.18   0.07   0.03   0.17   0.33 312.00   1.03   1.09
```

Bayes Factor for Covariate

Compute a Bayes factor

- H0: Slope parameter $\beta = 0$
- H1: Slope parameter $\beta \sim$ Cauchy(0, r) (with scale parameter r)
- Method: Savage-Dickey density ratio (Wagenmakers, 2010)
  - Bayes factor H1 vs. H0: prior devided by posterior density (at $\beta = 0$)

```
BayesFactorSlope(fit_regression,
                 parameter = "slope_d_continuous",
                 direction = ">",     # H1: positive slope parameter
                 plot = TRUE)         # plot Savage-Dickey density ratio
```




```
##                          BF_0>   BF_>0
## slope_d_continuous 0.2311668 4.32588
```

Between-Subject Comparisons: Similar to ANOVA

Between-subject designs: Assumptions about the covariance matrix Σ

1. Separate covariance matrix per condition: $\boldsymbol{\Sigma}_1, \boldsymbol{\Sigma}_2, \ldots$
   - See previous slides: `betweenSubjectMPT(fit1, fit2)`
2. Identical covariance matrix $\boldsymbol{\Sigma}$ across conditions
   - Similar to ANOVA: "pooled variance"
   - Manipulation only affects the mean parameters $\boldsymbol{\mu}$
   - ANOVA priors by Rouder & Morey (2012)

```
# fit all between-conditions jointly:
fit_between <- traitMPT(
                  htm_d, "2htm.csv",
                  covData = "covariates.csv",
                  predStructure = list("d ; discrete"),  # discrete predictor
                  predType = c("c","f")) # "c" =continuous; "f"=fixed-effects
```

```
round(fit_between$summary$groupParameters$factor[,-6], 2)
```

```
##                                 Mean   SD  2.5%   50% 97.5% n.eff Rhat R_95%
## factor_d_discrete[1]_group_a    0.14 0.16 -0.19  0.13  0.46   196 1.04  1.13
## factor_d_discrete[2]_group_b   -0.14 0.16 -0.46 -0.13  0.19   196 1.04  1.13
```

```
# get estimates for the group-specific MPT parameters
gmeans <- getGroupMeans(fit_between)
round(gmeans, 2)
```

```
##                                 Mean   SD 2.5%  50% 97.5% p(one-sided vs. overall)
```

Sensitivity/robustness analysis

Changing Priors

Define different priors

- Prior distributions in the latent-trait MPT necessary for:
  - Latent (probit-) mean μ
  - Latent (probit-) covariance matrix Σ: scaled inverse Wishart with
    - Prior matrix V
    - Degrees of freedom `df`
    - Scaling parameter ξ
- Example: We assume that guessing probabilities are around 50%

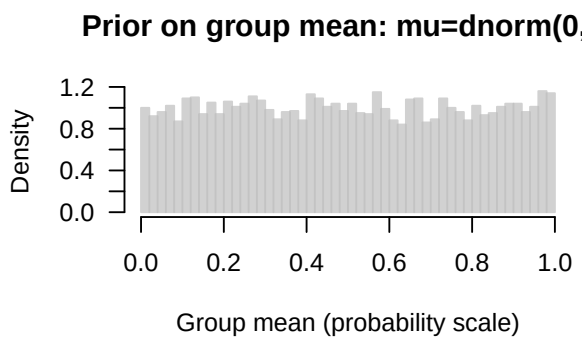
```
fit <- traitMPT(eqnfile="htm.txt", data="responses.csv",
                restrictions=list("dn=do"),

                mu = c(dn = "dnorm(0,1)",   # default prior
                       g = "dnorm(0,5)"),   # prior focused around 50% guessing
                xi = "dunif(0,2)",          # less disperson of MPT parameters
                V = diag(2),                # default
                df = 2 + 1)                 # default
```

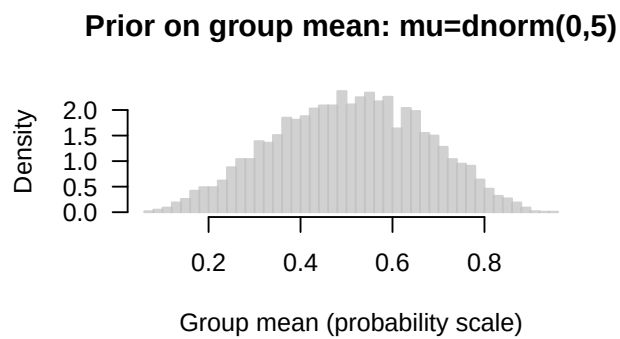
Understanding Priors

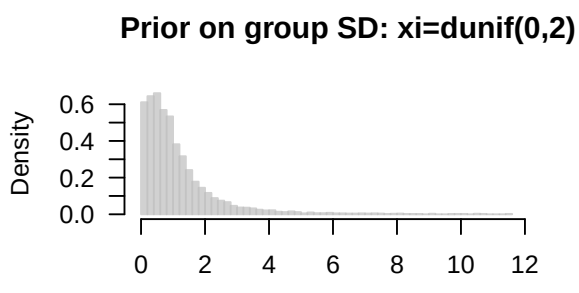
What do the priors actually mean?

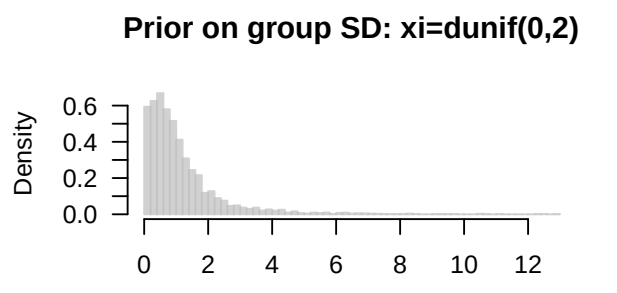
- Draw samples from the prior
- Plot mean/SD of MPT parameters

```
plotPrior(prior =
            list(mu = c(dn = "dnorm(0,1)",   # default prior
                        g =  "dnorm(0,5)"), # prior focused around 50%
                 xi="dunif(0,2)",           # smaller scale for group SD
                 V= diag(2), df = 3))       # default Wishart prior
```







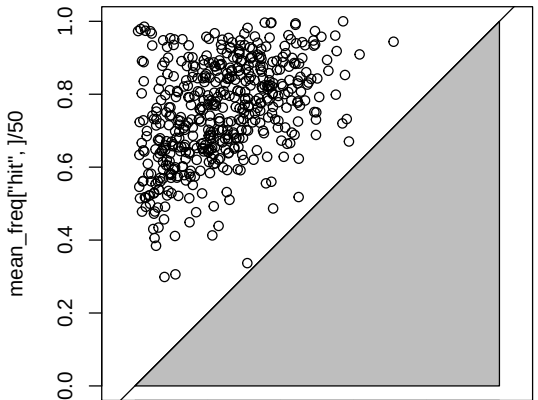


Prior Predictive Sampling

Prior predictive distribution

1. Draw samples from the prior
2. Draw new data (response frequencies)
3. Assess predicted data (e.g., plots or descriptive statistics)

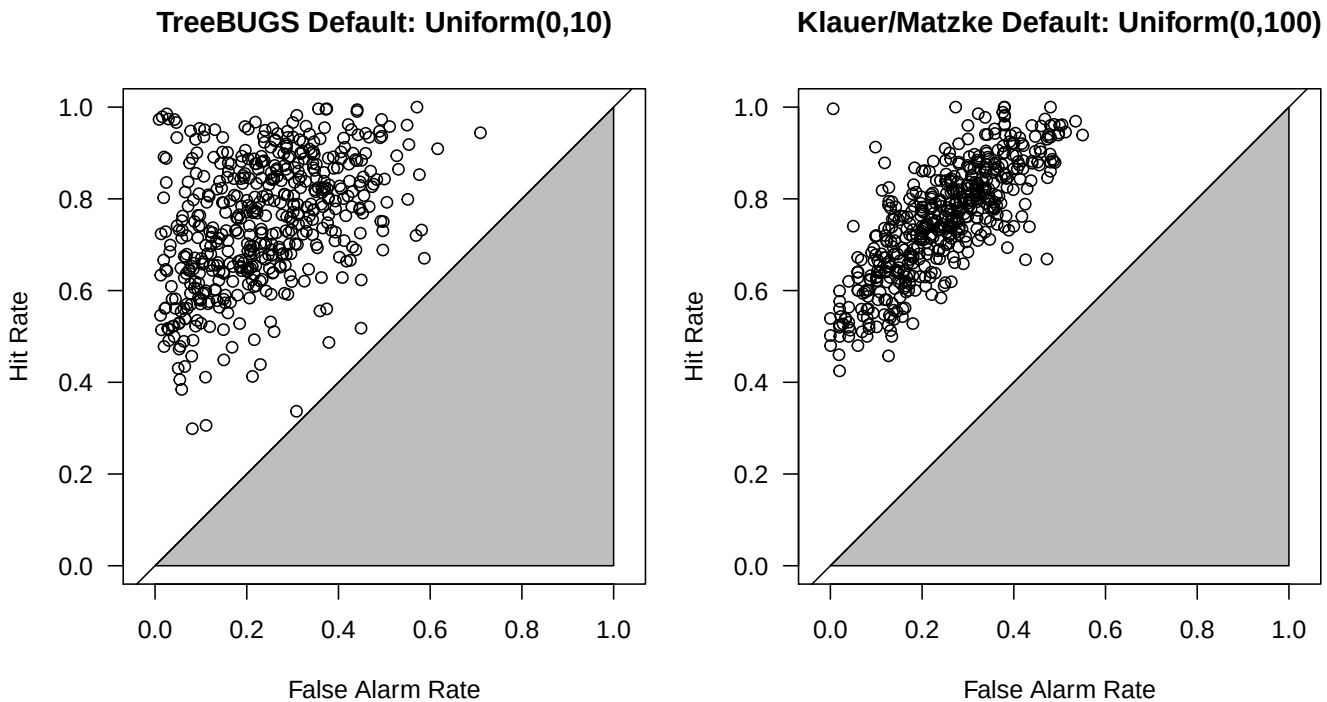
```
pp <- priorPredictive(prior = list(mu = "dnorm(0,1)", xi="dunif(0,10)",
                                   V=diag(2), df=2+1),
                      eqnfile = htm, restrictions = list("dn=do"),
                      numItems = c(target = 50, lure = 50),
                      N = 50,  M = 500)      # number of participants/samples

# compute and plot predicted values for the average hit/FA rates (in ROC space)
mean_freq <- sapply(pp, colMeans)
par(mar=c(4,5,.1, .1))
plot(mean_freq["fa",]/50, mean_freq["hit",]/50, asp = 1, xlim =0:1, ylim=0:1)
polygon(c(0,1,1), c(0,0,1), col = "gray")
abline(0, 1)
```



Prior Predictive Sampling

Excursion: Different default priors for the scale parameter ξ

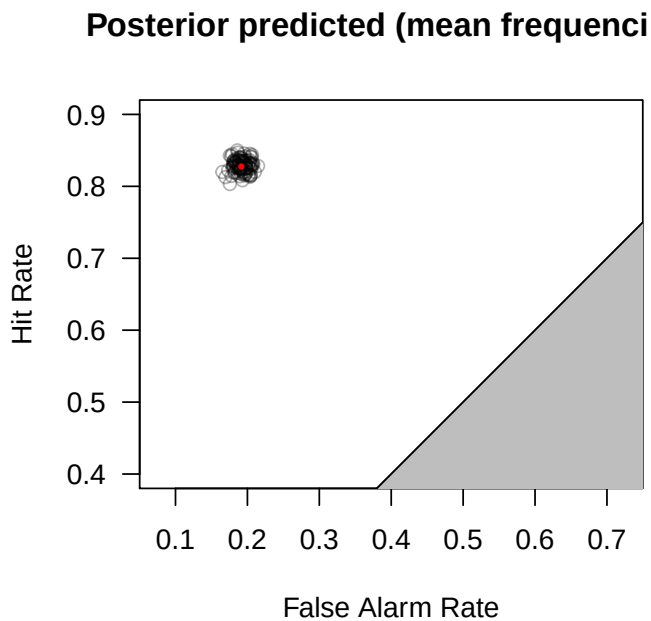
1. TreeBUGS (Heck et al., 2018): $\xi \sim \text{Uniform}(0, 10)$
2. Klauer (2010) and Matzke et al. (2015): $\xi \sim \text{Uniform}(0, 100)$



Posterior Predictive Distribution

- What data does the fitted model predict?
- Use posterior samples of the *parameters* to draw new samples of the *data* (i.e., predicted response frequencies)
  - Note: These are the basis of posterior-predictive checks (T1 and T2 statistics)

```
postpred <- posteriorPredictive(fit, M = 100, nCPU = 4)
```



Sensitivity and robustness analysis

- Assessing the impact of priors, estimate necessary sample size for specific analysis etc.
  1. Generate data from (correct or wrong) model
  2. Fit model
  3. If necessary: Replicate with a `for`-loop

```
# standard, fixed-effects MPT (generate data for one person)
sim <- genMPT(theta = c(d = .7, g = .5),
              numItems = c(target = 50, lure = 50),
              eqnfile = htm_d)

# hierarchical MPT (generate a complete table of frequencies)
sim2 <- genTraitMPT(N = 100, eqnfile = htm_d,
                    mean = c(d = .7, g = .5),
                    sigma = c(d = .4, g = .2),
                    rho = diag(2))
```

Appendix: Testing for Heterogeneity

- Test by Smith & Batchelder (2008)
  1. Test person heterogeneity assuming items homogeneity (χ^2)
  2. Test person heterogeneity under item heterogeneity (permutation bootstrap)

```
# A) chi^2 test
test <- testHetChi(freq = frequencies,
                   tree = c(cr="lure", fa="lure",
                            hit="target",miss="target"))
data.frame(test)
```

```
##      chisq df          prob
## 1 988.0927 98 4.893447e-147
```

```
# B) requires data in long format (variables: person / item / response)
# testHetPerm(data, tree, source = "person")
```

Appendix: MPT Versions Implemented in TreeBUGS

MCMC samplers available in TreeBUGS

- Fixed-effects ("standard") MPT: Gibbs sampler in `C++`
- Beta-MPT: `JAGS` and `C++`
- Latent-trait with extensions: `JAGS`
  - Combination of random- and fixed-effects parameters
  - Continuous and categorical covariates
  - Group-level structure: Independent normal distributions